**🔷 THEORY**

**📌 Problem Statement**

Given:

* A sequence of sorted keys k1<k2<⋯<knk\_1 < k\_2 < \dots < k\_nk1​<k2​<⋯<kn​
* Each key kik\_iki​ has an associated frequency pip\_ipi​, representing the probability of searching for that key

**Goal:** Build a Binary Search Tree (BST) that minimizes the **expected search cost**, i.e., the sum of (depth × frequency) for all nodes.

**📌 Optimal Binary Search Tree (OBST)**

An **Optimal BST** is a binary search tree where the expected search cost (considering search probabilities) is **minimum**.

**🔹Why Not Any BST?**

* A regular BST only considers key ordering.
* An OBST also considers **how often each key is accessed** (more frequent keys should be closer to the root).

**🔹Search Cost**

For a node at depth d with frequency f, cost = d × f.  
Total cost = sum of such values for all nodes.

**📌 Dynamic Programming Approach**

To build an OBST optimally, we use **Dynamic Programming (DP)**.

**Key Concepts:**

* cost[i][j]: Minimum cost of BST containing keys from kik\_iki​ to kjk\_jkj​
* rootIndex[i][j]: The index of the key chosen as root for subarray kik\_iki​ to kjk\_jkj​
* Total cost of a subtree rooted at krk\_rkr​:

cost=cost[i][r−1]+cost[r+1][j]+∑t=ijfreq[t]\text{cost} = \text{cost}[i][r-1] + \text{cost}[r+1][j] + \sum\_{t=i}^{j} \text{freq}[t]cost=cost[i][r−1]+cost[r+1][j]+t=i∑j​freq[t]

**🔷 ALGORITHM**

**🔧 Step-by-Step Algorithm**

text

CopyEdit

Input:

keys[]: sorted array of keys

freq[]: corresponding access frequencies

n: number of keys

Output:

Optimal BST with minimum expected search cost

1. Initialize 2D arrays:

- cost[0..n-1][0..n-1] = 0

- rootIndex[0..n-1][0..n-1] = 0

2. For i = 0 to n-1:

cost[i][i] = freq[i]

rootIndex[i][i] = i

3. For L = 2 to n: // L is subtree length

For i = 0 to n-L:

j = i + L - 1

cost[i][j] = ∞

fsum = sum(freq[i..j])

For r = i to j:

c = cost[i][r-1] + cost[r+1][j] + fsum

If c < cost[i][j]:

cost[i][j] = c

rootIndex[i][j] = r

4. Construct tree recursively using rootIndex[][]

5. Return the root of the tree

**🔧 Inorder Traversal**

Standard left-root-right traversal to verify the BST structure.